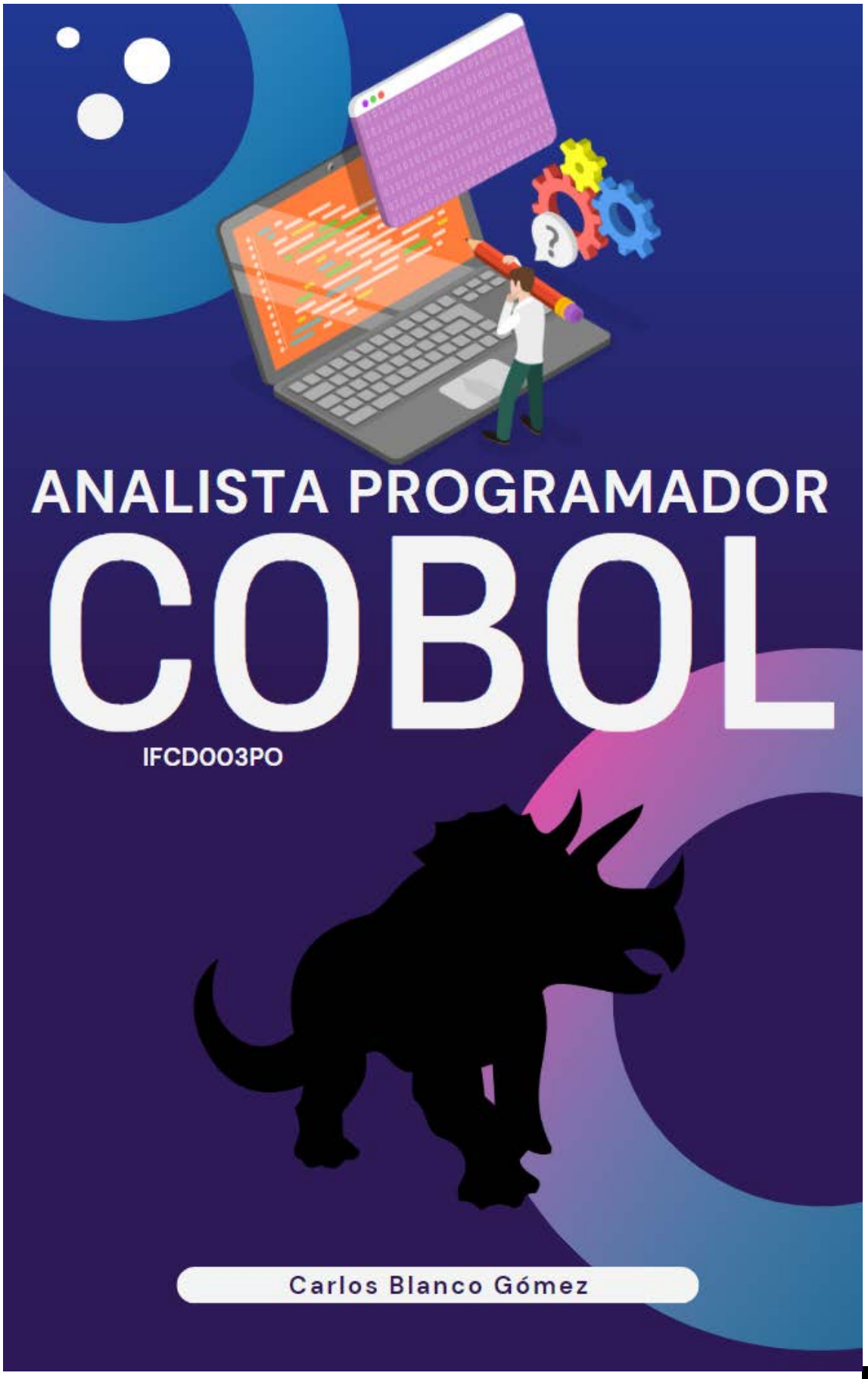
ANALISTA PROGRAMADOR
COBOL
IFCD003PO
Carlos Blanco Gómez

Contenido

Vamos a desglosar y explicar cada uno de los componentes de este programa que combina COBOL y Python, incluyendo el Dockerfile para crear la imagen Docker.

1. Programa COBOL (crud_cobol.cob)

Este programa en COBOL realiza operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre una base de datos utilizando comandos de sistema para ejecutar un script Python que maneja la lógica de la base de datos.

Secciones del Programa COBOL

1. **IDENTIFICATION DIVISION**: Esta es la sección donde se declara el nombre del programa.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CRUDOperations.
```

2. **ENVIRONMENT DIVISION**: Define el entorno en el que se ejecuta el programa, incluyendo archivos de entrada y salida.

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT RESULT-FILE ASSIGN TO 'result.txt'
```

```
        ORGANIZATION IS LINE SEQUENTIAL.
```

3. **DATA DIVISION**: Define las estructuras de datos utilizadas en el programa, incluyendo archivos y variables en la sección de trabajo.

```
DATA DIVISION.
FILE SECTION.
FD  RESULT-FILE.
01  RESULT-RECORD   PIC X(200).

WORKING-STORAGE SECTION.
01  CMD             PIC X(200).
01  WS-OPERATION    PIC X(10).
01  WS-ID           PIC 9(4).
01  WS-NAME         PIC X(100).
01  WS-EXIT         PIC X(3) VALUE 'NO '.
01  WS-EOF          PIC X(3) VALUE 'NO '.
01  EXIT-STATUS     PIC S9(4) BINARY.
```

4. **PROCEDURE DIVISION**: Contiene la lógica del programa, incluyendo un bucle principal para recibir comandos del usuario y ejecutar operaciones CRUD correspondientes mediante llamadas al script Python.

```
PROCEDURE DIVISION.
MAIN-LOGIC.
    PERFORM UNTIL WS-EXIT = 'YES'
        DISPLAY "operation: create, read, update, delete, exit"
        ACCEPT WS-OPERATION

        EVALUATE TRUE
            WHEN WS-OPERATION = "create"
                PERFORM CREATE-RECORD
            WHEN WS-OPERATION = "read"
                PERFORM READ-RECORDS
            WHEN WS-OPERATION = "update"
                PERFORM UPDATE-RECORD
            WHEN WS-OPERATION = "delete"
                PERFORM DELETE-RECORD
            WHEN WS-OPERATION = "exit"
                MOVE "YES" TO WS-EXIT
            WHEN OTHER
                DISPLAY "Invalid operation."
        END-EVALUATE

        DISPLAY "Operation completed."
    END-PERFORM

    DISPLAY "Exiting the program."
    STOP RUN.
```

Cada operación (crear, leer, actualizar, eliminar) se define como un procedimiento separado:

- **CREATE-RECORD**
- **READ-RECORDS**

- **UPDATE-RECORD**
- **DELETE-RECORD**

1. CREATE-RECORD

Este procedimiento se encarga de crear un nuevo registro en la base de datos.

Código:

```
CREATE-RECORD.
    DISPLAY "Enter name:"
    ACCEPT WS-NAME
    MOVE "python3 /app/crud_db.py create " TO CMD
    MOVE WS-NAME TO CMD (33:100)  *> Ajustando la posición según la
longitud del comando inicial
    DISPLAY "Command to execute: ", CMD
    CALL 'SYSTEM' USING CMD GIVING EXIT-STATUS
    DISPLAY "Exit status: ", EXIT-STATUS
    .
```

Descripción:

- **DISPLAY "Enter name:"**: Muestra un mensaje al usuario pidiendo que ingrese un nombre.
- **ACCEPT WS-NAME**: Acepta la entrada del usuario y la almacena en la variable `WS-NAME`.
- **MOVE "python3 /app/crud_db.py create " TO CMD**: Construye el comando que se va a ejecutar, iniciando con el comando básico para crear un registro.
- **MOVE WS-NAME TO CMD (33:100)**: Añade el nombre ingresado por el usuario al comando, ajustando la posición según la longitud del comando inicial.
- **DISPLAY "Command to execute: ", CMD**: Muestra el comando completo que se va a ejecutar.
- **CALL 'SYSTEM' USING CMD GIVING EXIT-STATUS**: Ejecuta el comando usando una llamada al sistema y almacena el estado de salida en `EXIT-STATUS`.
- **DISPLAY "Exit status: ", EXIT-STATUS**: Muestra el estado de salida de la ejecución del comando.

2. READ-RECORDS

Este procedimiento se encarga de leer todos los registros de la base de datos y mostrarlos al usuario.

Código:

```
READ-RECORDS.
    MOVE "python3 /app/crud_db.py read > result.txt" TO CMD
    CALL 'SYSTEM' USING CMD GIVING EXIT-STATUS
    OPEN INPUT RESULT-FILE
    PERFORM UNTIL WS-EOF = 'YES'
        READ RESULT-FILE INTO RESULT-RECORD
        AT END
            MOVE "YES" TO WS-EOF
        NOT AT END
            DISPLAY RESULT-RECORD
```

```
    END-PERFORM
    CLOSE RESULT-FILE
    MOVE "NO " TO WS-EOF
    DISPLAY "Exit status: ", EXIT-STATUS
    .
```

Descripción:

- **MOVE "python3 /app/crud_db.py read > result.txt" TO CMD**: Construye el comando para leer los registros y redirige la salida a un archivo `result.txt`.
- **CALL 'SYSTEM' USING CMD GIVING EXIT-STATUS**: Ejecuta el comando y almacena el estado de salida.
- **OPEN INPUT RESULT-FILE**: Abre el archivo `result.txt` para leer.
- **PERFORM UNTIL WS-EOF = 'YES'**: Bucle para leer el archivo hasta el final.
    - **READ RESULT-FILE INTO RESULT-RECORD**: Lee una línea del archivo.
    - **AT END**: Si se llega al final del archivo, establece `WS-EOF` a 'YES'.
    - **NOT AT END**: Si no es el final, muestra la línea leída.
- **CLOSE RESULT-FILE**: Cierra el archivo.
- **MOVE "NO " TO WS-EOF**: Reinicia la variable `WS-EOF`.
- **DISPLAY "Exit status: ", EXIT-STATUS**: Muestra el estado de salida de la ejecución del comando.

3. UPDATE-RECORD

Este procedimiento se encarga de actualizar un registro existente en la base de datos.

Código:

```
UPDATE-RECORD.
    DISPLAY "Enter ID to update:"
    ACCEPT WS-ID
    DISPLAY "Enter new name:"
    ACCEPT WS-NAME
    MOVE "python3 /app/crud_db.py update " TO CMD
    MOVE WS-ID TO CMD (32:4)
    MOVE " " TO CMD (36:1)
    MOVE WS-NAME TO CMD (37:100)
    DISPLAY "Command to execute: ", CMD
    CALL 'SYSTEM' USING CMD GIVING EXIT-STATUS
    DISPLAY "Exit status: ", EXIT-STATUS
    .
```

Descripción:

- **DISPLAY "Enter ID to update:"**: Muestra un mensaje pidiendo el ID del registro a actualizar.
- **ACCEPT WS-ID**: Acepta el ID ingresado por el usuario.
- **DISPLAY "Enter new name:"**: Muestra un mensaje pidiendo el nuevo nombre.
- **ACCEPT WS-NAME**: Acepta el nuevo nombre ingresado.
- **MOVE "python3 /app/crud_db.py update " TO CMD**: Construye el comando para actualizar un registro.
- **MOVE WS-ID TO CMD (32:4)**: Añade el ID al comando.
- **MOVE " " TO CMD (36:1)**: Añade un espacio entre el ID y el nombre.
- **MOVE WS-NAME TO CMD (37:100)**: Añade el nuevo nombre al comando.

- **DISPLAY "Command to execute: ", CMD**: Muestra el comando completo.
- **CALL 'SYSTEM' USING CMD GIVING EXIT-STATUS**: Ejecuta el comando y almacena el estado de salida.
- **DISPLAY "Exit status: ", EXIT-STATUS**: Muestra el estado de salida de la ejecución del comando.

4. DELETE-RECORD

Este procedimiento se encarga de eliminar un registro existente de la base de datos.

Código:

```
DELETE-RECORD.
    DISPLAY "Enter ID to delete:"
    ACCEPT WS-ID
    MOVE "python3 /app/crud_db.py delete " TO CMD
    MOVE WS-ID TO CMD (32:4)
    DISPLAY "Command to execute: ", CMD
    CALL 'SYSTEM' USING CMD GIVING EXIT-STATUS
    DISPLAY "Exit status: ", EXIT-STATUS
    .
```

Descripción:

- **DISPLAY "Enter ID to delete:"**: Muestra un mensaje pidiendo el ID del registro a eliminar.
- **ACCEPT WS-ID**: Acepta el ID ingresado por el usuario.
- **MOVE "python3 /app/crud_db.py delete " TO CMD**: Construye el comando para eliminar un registro.
- **MOVE WS-ID TO CMD (32:4)**: Añade el ID al comando.
- **DISPLAY "Command to execute: ", CMD**: Muestra el comando completo.
- **CALL 'SYSTEM' USING CMD GIVING EXIT-STATUS**: Ejecuta el comando y almacena el estado de salida.
- **DISPLAY "Exit status: ", EXIT-STATUS**: Muestra el estado de salida de la ejecución del comando.

Estos procedimientos permiten que el programa COBOL interactúe con la base de datos mediante un script Python, facilitando la gestión de los datos.

2. Script Python (crud_db.py)

Este script maneja las operaciones de base de datos utilizando SQLite. Define funciones para crear, leer, actualizar y eliminar registros.

Funciones del Script Python

1. **create(name)**: Inserta un nuevo registro en la base de datos.

   ```
   def create(name):
   ```

```
    conn = sqlite3.connect('/app/test.db')
    cursor = conn.cursor()
    cursor.execute("INSERT INTO greetings (name) VALUES (?)",
(name,))
    conn.commit()
    conn.close()
```

2. **read()**: Lee todos los registros de la base de datos.

```
def read():
    conn = sqlite3.connect('/app/test.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM greetings")
    rows = cursor.fetchall()
    conn.close()
    return rows
```

3. **update(id, name)**: Actualiza un registro específico en la base de datos.

```
def update(id, name):
    conn = sqlite3.connect('/app/test.db')
    cursor = conn.cursor()
    cursor.execute("UPDATE greetings SET name = ? WHERE id = ?",
(name, id))
    conn.commit()
    conn.close()
```

4. **delete(id)**: Elimina un registro específico de la base de datos.

```
def delete(id):
    conn = sqlite3.connect('/app/test.db')
    cursor = conn.cursor()
    cursor.execute("DELETE FROM greetings WHERE id = ?", (id,))
    conn.commit()
    conn.close()
```

5. **Bloque principal**: Maneja los argumentos de la línea de comandos y llama a las funciones correspondientes.

```
if __name__ == "__main__":
    print("Received arguments:", sys.argv)

    if len(sys.argv) < 2:
        print(f"Usage: {sys.argv[0]} <operation> [parameters]")
        sys.exit(1)

    operation = sys.argv[1]

    if operation == 'create':
        if len(sys.argv) != 3:
            print(f"Usage: {sys.argv[0]} create <name>")
            sys.exit(1)
        create(sys.argv[2])
    elif operation == 'read':
        rows = read()
```

```
        for row in rows:
            print(f"{row[0]}: {row[1]}")
    elif operation == 'update':
        if len(sys.argv) != 4:
            print(f"Usage: {sys.argv[0]} update <id> <name>")
            sys.exit(1)
        update(int(sys.argv[2]), sys.argv[3])
    elif operation == 'delete':
        if len(sys.argv) != 3:
            print(f"Usage: {sys.argv[0]} delete <id>")
            sys.exit(1)
        delete(int(sys.argv[2]))
```

3. Dockerfile

Este archivo define el entorno de Docker para ejecutar el programa COBOL junto con el script Python.

Pasos del Dockerfile

1. **Imagen base y actualización del sistema**:

   ```
   FROM ubuntu:latest

   RUN apt-get update && apt-get install -y \
       gnucobol \
       python3 \
       sqlite3 \
       wget
   ```

2. **Directorio de trabajo y copiado de archivos**:

   ```
   WORKDIR /app

   COPY crud_cobol.cob /app
   COPY crud_db.py /app
   COPY setup_db.sh /app
   ```

3. **Configuración y compilación**:

   ```
   RUN chmod +x /app/setup_db.sh
   RUN /app/setup_db.sh
   RUN cobc -x -o crud_cobol crud_cobol.cob
   ```

4. **Comando por defecto**:

   ```
   CMD ["./crud_cobol"]
   ```

Instrucciones para Construir y Ejecutar la Imagen Docker

1. **Construir la Imagen Docker**:

```
docker build -t cobol-hello-world7 .
```

2. **Ejecutar un Contenedor desde la Imagen**:

```
docker run --rm -it cobol-hello-world7
```

Con estas instrucciones, el programa COBOL y el script Python deberían trabajar juntos para gestionar la base de datos, utilizando Docker para facilitar el entorno de ejecución.

Para eliminar todas las imágenes y contenedores en Docker, puedes seguir estos pasos. Estos comandos son muy poderosos y eliminarán **todos** los contenedores e imágenes en tu sistema Docker, así que asegúrate de que realmente deseas hacerlo antes de proceder.

1. Eliminar Todos los Contenedores

Primero, detén y elimina todos los contenedores. Puedes hacerlo con los siguientes comandos:

Detener Todos los Contenedores
```
docker stop $(docker ps -aq)
```
Eliminar Todos los Contenedores
```
docker rm $(docker ps -aq)
```

2. Eliminar Todas las Imágenes

Después de haber eliminado todos los contenedores, puedes proceder a eliminar todas las imágenes:

Eliminar Todas las Imágenes
```
docker rmi $(docker images -q)
```

Comandos Explicados

- `docker ps -aq`: Lista todos los contenedores (IDs solamente).
- `docker stop $(docker ps -aq)`: Detiene todos los contenedores listados.
- `docker rm $(docker ps -aq)`: Elimina todos los contenedores listados.
- `docker images -q`: Lista todas las imágenes (IDs solamente).
- `docker rmi $(docker images -q)`: Elimina todas las imágenes listadas.

Eliminar Volúmenes y Redes (Opcional)

Si también deseas limpiar volúmenes y redes, puedes usar estos comandos:

Eliminar Todos los Volúmenes
```
docker volume rm $(docker volume ls -q)
```
Eliminar Todas las Redes (excepto las redes predeterminadas)
```
docker network rm $(docker network ls | grep -v "bridge\|host\|none" |
awk '{if (NR>1) print $1}')
```

Forzar la Eliminación

Si encuentras problemas de dependencias o si quieres forzar la eliminación de contenedores e imágenes, puedes agregar el parámetro `-f` a los comandos `docker rm` y `docker rmi`. Por ejemplo:

```
docker rm -f $(docker ps -aq)
docker rmi -f $(docker images -q)
```

Con estos pasos, deberías haber eliminado todos los contenedores, imágenes, volúmenes y redes no deseadas en tu sistema Docker.